relative: If relative has the value 1 the least squares method is applied with respect to the relative error as described above, else with respect to the absolute error.

drumomega, drumR, drumI, druma, drumb, drumc:

Drumplace values assigned to the first elements of the arrays ω_k, R_k, I_k, [A], [B], [X]. These values must be chosen to permit space enough for the respective arrays. It is noticed, that ω_k, R_k and I_k have each m elements and [A] (p+q+1) elements, while [B] and [X] have each p+q+1 elements.

drumA: The drum section beginning with this drumplace value is to contain the greatest number of elements of the following two: 4(p+q+1) or 4 m.

reals: error:

The desired error according to the above described error criterion for stopping the iterations.

labels: singular:

This label originates from the procedure DRUMCROUT 2 (by O. Lang Rasmussen) which is used to solve the matrix equation. The procedure goes to singular when the determinant of matrix [A] is zero.

Besides the existence in the fast memory of the actual parameters corresponding to the above mentioned formal parameters, the procedure requires the presence in the drum store of the arrays ω_k, R_k and I_k at the positions determined by the respective drumplace numbers as explained above.

As to the results of the work of the procedure, the following is important:

The calculated coefficients of the polynomials of the fit function are stored on drum at the drumplace drumc. The order of the coefficients is:

$a_0, a_1, a_2, \ldots, a_p, b_1, b_2, \ldots, b_q$ when the fit function is written on the form:

$$G(s) = \frac{a_0 + a_1 s + a_2 s^2 + \ldots + a_p s^p}{1 + b_1 s + b_2 s^2 + \ldots + b_q s^q}$$

The real and imaginary parts of $G(\omega_k)$, $k = 1, 2, \ldots, m$, are also calculated and stored on drum at the drumplace drumA. The order is: $RG_1, RG_2, \ldots, RG_m, IG_1, IG_2, \ldots, IG_m$.

Notice:

The quantity ,,error,, which was specified by the user, after the application of the procedure alters its value to the actual error of the resulting fit function. Similarly, the integer N, maximum number of iterations desired, alters its value to the actual number of iterations performed.

The arrays ω_k, R_k and I_k originally placed on drum by the user, are unviolated during the calculations.

Computing time

The computing time is propertional to the number of iterations performed and increases according to the second power of the quantity p+q+1. With p+q+1 = 14 and 10 iterations the computing time has been measured to about 10 minutes.

Remarks to the Algorithm:

If the relative error in a certain fit point tends to infinity (i.e. exceeds 10^{70}) the value 10^{70} is substituted. Similarly the weighting factor $1/|P(\omega_k)|^2$ is bounded within the limit 10^{70}.

It is noted that the case of the determinant of [A] being zero is closely related to the degree of indeterminacy of the coefficients of the fit function (f.ex. if the known function is a real constant, all choices of p an q except $p = 0$, $q = 0$ lead to indeterminacy).

It may be of interest to illustrate the use of the parameter No by an example:

Let J[1:L] be an integer array containing the iteration numbers after which printing of results is desired. The printing is supposed to take place at label W.

```
Algorithm: No := 0; a:= error;
           for i:= 1 step 1 until L do begin
           N := J[i];
           COMPFIT ( .... , N,No, '.....);
           W: if error < a then goto Out;
           error := a; No:= J[i]; end of i;
           Out:
```

This algorithm is used in the program PROFIT (see ref. 3).

References:

1) E. C. Levy: Complex - Curve Fitting.
IRE Transactions on Automatic Control, May 1959.

2) C.K. Sanathanan and J. Koerner:
Transfer Function Synthesis as a Ratio of Two Complex Polynomials.
IEEE Transactions on Automatic Control, May 1963.

3) N. Kjær-Pedersen: PROFIT. A program for Complex - Curve Fitting. P - 201.
Risø - M - 124, Sep. 1964.

The Algorithm:

```
procedure COMPFIT (m,p,q,N,No,error,relative,drumomega,drumR,drumI,druma,drumb,
                   drumc,drumA,singular);
integer m,p,q,N,No,relative,drumomega,drumR,drumI,druma,drumb,drumc,drumA;
real error;
label singular;
A1: begin
integer i,j,k,n,t;
real epsa,epsph,determinant,y,a,b; y := (3.14159/180) ↑2;
n:=No;
Iteration:
B1: begin comment The arrays lambda,S,T,and U are calculated and stored on
drum;
real array fitpoint,omega,R,I,W[1:m];
comment The array fitpoint is used as a means of gradually raising the power
of omega. The elements of W are the weighting factors;
drumplace:= drumomega; from drum (omega);
if n = 0 then
for k:= 1 step 1 until m do W[k]:= 1 else

begin drumplace:= drumA; from drum (R); from drum(I); from drum(W); end;
drumplace:= drumR; from drum(R);
drumplace:= drumI; from drum(I);
C1: begin
real array lambda[0:2×p];
for k:= 1 step 1 until m do
fitpoint[k]:= 1;
for i:= 0 step 1 until 2×p do begin
lambda[i]:= 0;
```

```
for k:= 1 step 1 until m do begin
if i=0 then fitpoint[k]:=1 else
fitpoint[k]:= omega [k]xfitpoint[k];
lambda[i]:= lambda[i]+fitpoint[k]xW[k] end; end;
drumplace:= drumA; to drum(lambda);
end of C1;

C2: begin
real array S[0:p+q];
for k:= 1 step 1 until m do
fitpoint[k]:= 1;
for i:= 0 step 1 until p+q do begin
S[i]:= 0;
for k:= 1 step 1 until m do begin
if i=0 then fitpoint[k]:=1 else
fitpoint[k]:= omega[k]xfitpoint[k];
S[i]:= S[i]+fitpoint[k]xW[k]xR[k] end; end;
to drum (S);
end of C2;

C3: begin
real array T[0:p+q];
for k:= 1 step 1 until m do
fitpoint[k]:= 1;
for i:= 0 step 1 until p+q do begin
T[i]:= 0;
for k:= 1 step 1 until m do begin
if i=0 then fitpoint[k]:=1 else
fitpoint[k]:= omega[k]xfitpoint[k];
T[i]:= T[i]+fitpoint[k]xW[k]xI[k] end; end;
to drum (T);
end of C3;

C4: begin
real array U[0:2xq];
for k:= 1 step 1 until m do
fitpoint[k]:= 1;
for i:= 0 step 1 until 2xq do begin
U[i]:= 0;
for k:= 1 step 1 until m do begin
if i=0 then fitpoint[k]:=1 else
fitpoint[k]:= omega[k]xfitpoint[k];
U[i]:= U[i]+fitpoint[k]xW[k]x(R[k] ↑ 2+I[k] ↑ 2) end; end;
to drum (U);
end of C4;

end of B1;
B2: begin comment The system of p+q+1 linear equations is, row after row,
constructed and stored on drum;
real array row[0:p+q],lambda[0:2xp],S,T[0:p+q],U[0:2xq];
integer t;
drumplace:= drumA; from drum (lambda); from drum(S);from drum(T);from drum(U);
drumplace:= drumb;
for i:= 0 step 1 until p do
if entier (i/2) = i/2 then row[i]:= -S[i] else row[i]:=-T[i];
for i:= 1 step 1 until q do
if entier (i/2) = i/2 then row[p+i]:= -U[i] else row[p+i]:= 0;to drum(row);
```

```
drumplace:= druma;
for i:= 0 step 1 until p do begin
for j:= 0 step 1 until p do begin
if entier (i/2) = i/2 ∧ entier (j/2) ≠ j/2 then row [j] := 0;
if entier (i/2) = i/2 ∧ entier (j/2) = j/2 then begin
t:= (j+2)/2; row [j] := (-1) ↑ txlambda[i+j]; end;
if entier (i/2) ≠ i/2 ∧ entier (j/2) = j/2 then row [j] :=0;
if entier (i/2) ≠ i/2 ∧ entier (j/2) ≠ j/2 then begin
t:= (j+1)/2; row [j] :=(-1) ↑ txlambda[i+j]; end;

end;
for j:= 1 step 1 until q do begin
if entier (i/2) = i/2 ∧ entier (j/2) ≠ j/2 then begin
t:= (j+1)/2; row [p+j] :=(-1) ↑ txT[i+j]; end;
if entier (i/2) = i/2 ∧ entier (j/2) = j/2 then begin
t:= (j/2); row [p+j] :=(-1) ↑ txS[i+j]; end;
if entier (i/2) ≠ i/2 ∧ entier (j/2) = j/2 then begin
t:= (j/2); row [p+j] :=(-1) ↑ txT[i+j]; end;
if entier (i/2) ≠ i/2 ∧ entier (j/2) ≠ j/2 then begin
t:= (j-1)/2; row [p+j] :=(-1) ↑ txS[i+j]; end;

end;
to drum (row); end;
for i:= 1 step 1 until q do begin
for j:= 0 step 1 until p do begin
if entier (i/2) = i/2 ∧ entier (j/2) ≠ j/2 then begin
t:= (j+1)/2; row [j] :=(-1) ↑txT[i+j]; end;
if entier (i/2) = i/2 ∧ entier (j/2) = j/2 then begin
t:= (j+2)/2; row [j] :=(-1) ↑txS[i+j]; end;
if entier (i/2) ≠ i/2 ∧ entier (j/2) = j/2 then begin
t:= (j/2); row [j] :=(-1) ↑txT[i+j]; end;
if entier (i/2) ≠ i/2 ∧ entier (j/2) ≠ j/2 then begin
t:= (j+1)/2; row [j] :=(-1) ↑txS[i+j]; end;
end;
for j:= 1 step 1 until q do begin
if entier (i/2) = i/2 ∧ entier (j/2) ≠ j/2 then row [p+j] :=0;
if entier (i/2) = i/2 ∧ entier (j/2) = j/2 then begin
t:= (j/2); row [p+j] :=(-1) ↑txU[i+j]; end;
if entier (i/2) ≠ i/2 ∧ entier (j/2) = j/2 then row [p+j] :=0;
if entier (i/2) ≠ i/2 ∧ entier (j/2) ≠ j/2 then begin
t:= (j-1)/2; row [p+j] :=(-1) ↑ txU[i+j];end;
end;

to drum (row); end;
end of B2;

B3: begin comment The system of equations is normalized by
dividing each equation with the length of the left side vector;
real array row, norm[0:p+q];
real fix;
drumplace:= druma;
for i:= 0 step 1 until p+q do begin
fix:= drumplace;
from drum(row);
norm[i]:= 0;
for j:= 0 step 1 until p+q do
norm[i]:= norm[i]+row[j] ↑2;
norm[i]:= sqrt(norm[i]);
for j:= 0 step 1 until p+q do
```

```
if norm[i] ≠ 0 then
row[j]:= row[j]/norm[i];
drumplace:= fix;
to drum(row); fix:= drumplace;
drumplace:= drumb;
from drum(row);
if norm[i] ≠ 0 then row[i]:= row[i]/norm[i];
drumplace:= drumb;
to drum(row);
drumplace:= fix;
end;
end of B3;

B4: begin comment The equations are solved by means of the procedure DRUMCROUT2
and the results, which are the desired coefficients of numerator and denominator
polynomials of fit function, are stored on drum;
comment AEK february 7th. 1963 - the following procedure is a
drum version of SA - 10 it solves a system of linear equations Ay = b
as descreibed in SA - 26;

real procedure INNERPRODUCT(u,v,k,s,f);
value s,f;
integer k,s,f;
real u,v;

begin
real h;
h := 0;
for k := s step 1 until f do h := h + u × v;
INNERPRODUCT := h
end INNERPRODUCT;

procedure DRUMCROUT 2(ta,tb,ty,n,det);

value ta,tb,ty,n;
integer ta,tb,ty,n;
real det;

begin

integer tk,td,k,i,j,imax,timax,p;

real temp,quot;
real array A1,A2,b,y[1:n];

det := 1.0; tk := ta; drumplace := tb; td:= from drum(b);
for k := 1 step 1 until n do
L1 : begin drumplace := ta;
     for p := 1 step 1 until k-1 do
L2 : begin comment transport k-1 elements from column k to fast memory storage;
     from drum(A1); A2[p] := A1[k]
end L2;

temp := 0; drumplace := tk;
for i := k step 1 until n do

L3 : begin
from drum(A1); drumplace := drumplace - td;
A1[k] := A1[k]-INNERPRODUCT(A1[p],A2[p],p,1,k-1);
```

```
if abs (A1[k])>temp then
L4 : begin temp := abs(A1[k]); timax := drumplace; imax := i
end L4;
to drum(A1)
end L3;

comment we have found that element k in row imax is the largest pivot
element in column k, we now interchange rows k and imax;
if imax ≠ k then

L5 : begin det := -det;
drumplace := tk; from drum(A1);

drumplace := timax; from drum(A2);
drumplace := tk; to drum(A2);
drumplace := timax; to drum(A1);
temp := b[k]; b[k] := b[imax]; b[imax] := temp
end L5;

comment end of row interchange, next follows elimination;
drumplace := tk; from drum(A1);
if A1[k] = 0 then goto singular;
quot := 1.0/A1[k];
for i := k+1 step 1 until n do
L6 : begin

from drum(A2); A2[k] := quot × A2[k];
drumplace := drumplace -td; to drum(A2)
end L6;

drumplace := ta;
for p := 1 step 1 until k-1 do

L7 : begin
from drum(A2);
for j := k+1 step 1 until n do
A1[j] := A1[j] - A1[p] × A2[j];
end L7;
b[k] := b[k] - INNERPRODUCT (A1[p],b[p],p,1,k-1);
drumplace := tk; to drum(A1); tk := drumplace
end L1;

comment end of triangularization now comes back substitution;

for k := n step -1 until 1 do

L8 : begin
drumplace := drumplace -td; from drum(A1);
drumplace := drumplace -td; det := A1[k] × det;
y[k]:=(b[k]-INNERPRODUCT(A1[p],y[p],p,k+1,n))/A1[k]
end L8;

drumplace := ty; to drum(y);
end DRUMCROUT 2;
DRUMCROUT 2(druma,drumb,drumc,p+q+1,determinant);
end of B4;
```

```
B5: begin comment Real and imaginary part of numerator and denominator of fit
function are calculated for each fit point and stored on drum;
real array fitpoint,A,B,C,D[1:m],row[0:p+q];
comment The array fitpoint is here used to contain the values of omega ↑2;
drumplace:= drumomega; from drum(fitpoint);
drumplace:= drumc; from drum(row);
for k:= 1 step 1 until m do begin
fitpoint[k]:= fitpoint[k] ↑ 2;
if entier(p/2)= p/2 then j:= p else j:= p-1;
b:= row[j];
if j > 0 then
for i:= j-2 step -2 until 0 do begin
a:= - fitpoint[k]xb;
b:= a+row[i]; end; A[k]:= b;
if entier (p/2) = p/2 then j:=p-1 else j:= p;
if j= -1 then b:= 0 else
b:= row[j];
if j > 1 then
for i:=j -2 step -2 until 1 do begin
a:= -fitpoint[k] xb;
b:= a+row[i]; end; B[k]:= bxsqrt(fitpoint[k]);
if entier (q/2) = q/2 then j:= q else j:= q-1;
b:= row[p+j];if j=0 then b:=0;
if j > 2 then
for i:= j-2 step -2 until 2 do begin
a:= - fitpoint[k]xb;
b:= a+ row[p+i]; end; C[k]:= 1-fitpoint[k]xb;
if entier(q/2)= q/2 then j:= q-1 else j:= q;
if j= -1 then b:= 0 else
b:= row[p+j];
if j > 1 then
for i:= j-2 step -2 until 1 do begin

a:= -fitpoint[k]xb;
b:= a +row[p+i]; end; D[k]:= bxsqrt(fitpoint[k]); end k;
drumplace:= drumA; to drum(A);to drum(B);to drum(C);to drum(D);
end of B5;

B6: begin comment Real and imaginary part of fit function are calculated for
each fit point, compared with input values and stored on drum. Weighting factors
for next iteration are calculated and stored on drum. Criterion for stopping the
iterations is applied;
real array A,B,C,D,R,I[1:m];
real ea,ep;
drumplace:= drumA; from drum(A); from drum(B); from drum(C); from drum(D);
drumplace:= drumR; from drum(R);
drumplace:= drumI; from drum(I);
t:= 0; ea := ep := 0;

for k:= 1 step 1 until m do begin
epsa:= (A[k]xC[k]+B[k]xD[k])/(C[k] ↑2+D[k] ↑ 2) -R[k];
epsph:= (B[k]xC[k]-A[k]xD[k])/(C[k] ↑2+D[k] ↑ 2)-I[k];
C[k]:= (if relative = 1 then(if A[k]↑2+B[k]↑2<10-70
then 1070 else 1/(A[k]↑2+B[k]↑2)) else
1/(C[k]↑2+D[k]↑2));
A[k]:= R[k]+epsa; B[k]:= I[k]+epsph;
```

```
if (sqrt(A[k]↑2 + B[k]↑2)×10-70 < sqrt(R[k]↑2 + I[k]↑2)) ∧ (sqrt(R[k]↑2 +
                                                              I[k]↑2)≠0) then
epsa:= abs(sqrt(R[k] ↑2+I[k] ↑2)-sqrt(A[k] ↑ 2+B[k] ↑2))/sqrt(R[k] ↑2+I[k] ↑2) else
epsa:= 10↑70;
epsph:= abs((if R[k] ≠ 0 ∧ abs(I[k]×10-154) < abs(R[k]) then
        ( if R[k] > 0 then arctan (I[k]/R[k]) else
         (if I[k] = 0 then 3.14159 else(if I[k]>0 then arctan(I[k]/R[k]) + 3.14159
else arctan(I[k]/R[k]) - 3.14159)))else sign(I[k]) ×3.14159/2)
        -(if A[k] ≠ 0 ∧ abs(B[k]×10-154) < abs(A[k]) then
        (if A[k] > 0 then arctan(B[k]/A[k]) else
(if B[k] = 0 then 3.14159 else (if B[k] > 0 then arctan(B[k]/A[k]) + 3.14159
else arctan(B[k]/A[k]) - 3.14159))
else sign (B[k])×3.14159/2))
× 180/3.14159;
ea:= ea + epsa↑2; ep:= ep + epsph↑2;
end;

n:= n+1; drumplace:= drumA; to drum(A); to drum(B); to drum(C);
if n= N ∨(ea + y × ep)/m < error ↑ 2 then goto Stop else goto Iteration;
Stop:
N:= n;
error := sqrt((ea+y×ep)/m);
end of B6;
end of A1 and procedure;
```

N. Kjær-Pedersen.

DANISH ATOMIC ENERGY COMMISSION
Research Establishment Risø
Reactor Physics Section
Computer Group

September 1964
SA - 67
25 copies

Solution of a system of linear equations

ALGOL - procedure CROUT 4.

1. Scope.

The real procedure CROUT 4 solves a system of linear equations Ay = b with different right hand sides. A is a square matrix of order n, y and b are column vectors of dimension n. The problem is equivalent with solving a matrix equation A×Y=B where Y and B are matrices with n rows and m columns. The value of CROUT 4 is the determinant of the coefficient matrix A.

2. Method.

The solution is based on Crouts method with pivoting such as described for procedure CROUT in SA 10. It is therefore not repeated here. The main difference between CROUT and CROUT 4 is that the real procedure INNERPRODUCT in CROUT here is replaced by an innerproduct procedure written in machine code, which results in a considerably increased speed of computation. Furthermore an equilibration procedure has been incorporated in CROUT 4. The real procedure CROUT 4 is an extension of CROUT 3 in that respect that it can process several right hand sides in contradistinction to CROUT 3 which only can solve a linear system with one right hand side.

3. Use of the procedure.

The procedure consists of two parts, the algol procedure and a machine code. The procedure call shall be of the form

CROUT 4(n,A,drum,length,exit,factor,pivot,repeat);

where the parameters are:

repeat: declared as boolean. The first time CROUT 4 is called this variable must have the value false. In all the following calls the variable shall have the value true.

n: declared as integer is the order of the coefficient matrix.

A: declared as real array A[1:n,1:n+1] contains on the first call the coefficient matrix in A[1:n,1:n] and the first right hand side to be processed in A[1:n,n+1]. On the later calls the other right hand sides to be processed are stored consecutively in A[1:n,n+1].
On exit of each call the solution vector is stored in A[1:n,n+1]. During the first call the procedure performs a triangularization of the coefficient matrix. The triangularized matrix is stored in A[1:n,1:n] where it shall remain during the following calls of the procedure.

drum: declared as integer contains the value of the standard variable drumplace, when the innerproduct in code is stored on the drum by the standard procedure gierdrum. (see below).

length: declared as integer contains the number of machine-words occupied by the machine code as supplied by the standard procedure gierdrum. (see below).

exit: a label to which a jump is made from CROUT 4 when the coefficient matrix is singular.

factor: declared as real array factor [1:n] contains factors used in the equilibration process. These factors are computed and stored by the procedure during the first call with the boolean variable repeat having the value false, and they shall remain in this array during the following calls.

pivot: declared as integer array pivot [1:n] contains row indices originating from the triangularization of the coefficient matrix stored in A[1:n, 1:n]. These row indices are stored by the procedure during the first call with the boolean variable repeat having the value false, and they shall remain in this array during the following calls.

The innerproduct in code must be stored on the drum before the call of CROUT 4. This shall be done by the program and can be performed in the following way.

Assuming that the variable drum has been assigned some value of drumplace, one writes e.g. in the beginning of the program:

```
if kbon then
begin
writetext(§<
message to operator: innerproduct-code in tapereader§);
typechar;
drumplace:= drum;
gierdrum(§<innerp§,length);
writetext(§<
message to operator: set KB:= N§);
typechar;
end;
writetext(§<
m essage to operator: datatape in tapereader§);
typechar;
```

When the program is started with KB=L after the translation is completed GIER stops ready for input of innerproduct-code. After input of the machine code GIER stops for switching off the KB-button. Started again it stops ready for input of data after which the computation begins. The first parameter, §<innerp§, in the procedure call gierdrum (§<innerp§,length) is a code identification for innerproduct, which must be written exactly in this way. To the second parameter, length, is assigned the number of machine-words in the code.

The determinant of the coefficient matrix is obtained as the value of CROUT 4 during to first call of procedure.

4. References.

1. Communications of the ACM, 4, 1961, pp.176-77.
2. SA - 10, CROUT ALGOL procedure by O. Lang Rasmussen 3/6 - 1962
3. SA - 14/1, CROUT 2 ALGOL procedure by O. Lang Rasmussen 5/7 - 1964
4. SA - 64, CROUT 3 ALGOL procedure by O. Lang Rasmussen 8/7 - 1964
5. SM - 9, INNERPRODUCT procedure by O. Lang Rasmussen 10/9 - 1964

5. Algorithm.

```
real procedure CROUT 4(n,A,drum,length,exit,factor,pivot,repeat);
value n,drum,length;integer n,drum,length;
real array A,factor; integer array pivot;boolean repeat;label exit;
begin
integer i,j,imax,k,p1,p2,p3,p4,gem;
real t,q,det,detfactor;
boolean array code[1:length];
boolean entry;
gem:= drumplace;
drumplace:=drum;
fromdrum(code);
gierproc(code[2],A,p1,p2,p3,p4,entry);
detfactor:= det := 1;
if repeat then goto L1;
for i:= 1 step 1 until n do
begin comment equilibrateA[1:n,1:n+1];
q:= 0;
for j:= 1 step 1 until n do
begin t:= abs(A[i,j]); if t > q then q:= t end;
if q= 0 then goto exit;
factor[i] := t:= 2 ↑ (-entier(ln(q)/0.693147181+1));
for j:= 1 step 1 until n+1 do A[i,j]:= A[i,j]xt;
detfactor:= detfactorxt;
end equilibration;
p3:=1;
for k := 1 step 1 until n do
begin comment triangularization starts;
t := 0;
p2:=k;p4:=k-1;
for i := k step 1 until n do
begin
p1:=i;
A[i,k] := A[i,k] - gier(entry);
if abs(A[i,k]) > t then
begin t := abs(A[i,k]); imax := i end;
end;
pivot[k]:=imax;
comment the largest pivot element A[imax,k] in column k is found;
if imax ≠ k then
begin comment interchange rows k and imax;
det := -det;
for j := 1 step 1 until n+1 do
begin t := A[k,j]; A[k,j] := A[imax,j]; A[imax,j] := t end;
end interchange of rows;
if A[k,k] = 0 then goto exit;
q := 1/A[k,k];
for i := k + 1 step 1 until n do
A[i,k] := q × A[i,k];
p1:=k;
```

```
for j := k + 1 step 1 until n+1 do
begin
p2:=j;
A[k,j] := A[k,j] - gier(entry)
end;
det:=A[k,k]xdet;
end triangularization;
goto L2;

L1:
for i := 1 step 1 until n do A[i,n+1] := A[i,n+1]xfactor[i];
p2:=n+1;p3:=1;
for k:=1 step 1 until n do
begin
t:= A[pivot[k],n+1]; A[pivot[k],n+1]:= A[k,n+1]; A[k,n+1]:=t;
p1:=k; p4:=k-1;
A[k,n+1]:= A[k,n+1] - gier(entry)
end;

L2:
p2:=n+1; p4:=n;
for k := n step -1 until 1 do
begin comment backsubstitution;
p1:=k; p3:=k+1;
A[k,n+1] :=(A[k,n+1] - gier(entry))/A[k,k];
end backsubstitution;

CROUT 4:= det/detfactor;
drumplace:= gem;
end CROUT 4;
```

6. Innerproduct in code.

```
[innerproduct in code]
b a12
m
qq 10.3+39.9+41.15+53.21+37.27+37.33+57.39;{<innerp}
r
[first entry]
                arn a3          D           ;Radr:=jump adress
                ar a1           ,gr(p9)     ;entry:=hv-instruction
                arn p4          ,gr a1      ;array description
                arn(a1)         ,tkfm1      ;Radr:=c2=n+2
                ga a4           ,tkm-10     ;a4[adr]:=c2
                gt a8                       ;a8[tæl]:=c2
                arn(a1)         t-1         ;
                ar (a1)         t-1         ;
                tkm 30          ,gr a1      ;a1[adr]:=array length+constant
                arn p4          ,tkm 10     ;Radr:=adr[last element]+1
                sr a1           ,gr a1      ;a1[adr]:=basisadr-constant
                it(p5)          ,pa a3      ;a3[adr]:=p1
                it(p6)          ,pa a10     ;a10[adr]:=p2
                it(p7)          ,pa a11     ;a11[adr]:=p3
                it(p8)          ,pa a12     ;a12[adr]:=p4
                hr s1                       ;return to ALGOL

[later entries]

a3:             arfnm0          ,tkfm1      ;Radr:=p1
                xr                          ;Madr:=p1
a4:             mknm0           D           ;R18:=p1×c2
                tkm9            ,ar a1      ;Radr:=p1×c2+basisadr-constant
                ga a7                       ;a7[adr]:=Radr
a10:            arfnm0          ,tkfm1      ;Radr:=p2
                ar a1           ,ga a8      ;a8[adr]:=basisadr+p2-constant
a11:            arfnm0          ,tkfm1      ;
                ga a2                       ;a2[adr]:=p3
                srm1            DX          ;Radr:=p3-1,Madr:=Radr
                mkn(a4)         D           ;R18:=(p3-1)×c2
                tkm9            ,ac a8      ;a8[adr]:=basisadr+p2+(p3-1)×c2-constant
a12:            arfnm0          ,tkfm-9     ;Rtæl:=p4
                gt a5           ,gp a6      ;a5[tæl]:=p4,a6[adr]:=p
a2:             ppm0            ,grn a9     ;p:=p3,a9:=0
a5:             bs p            t0          ;if p>p4 then a6 else a7
a6:             ppm0            ,hr s1      ;return
a7:             arfn p0                     ;RF:=A[p1,p]
a8:             mkfm0           t0          ;RF:=A[p1,p]×A[p,p2]
                arf a9          ,grf a9     ;accumulate product
                pp p+1          ,hv a5      ;p:=p+1,jump
a1:             hv                          ;hv-instruction,storage for basisadr-constant
a9:             qq                          ;working cell
e
s
```

O. Lang Rasmussen.

DANISH ATOMIC ENERGY COMMISSION
Research Establishment Risø
Reactor Physics Department
Computer Group

April 1967
SA - 69/1
50 copies

Solution of a system of linear equations

ALGOL-procedure: DRUMCROUT 4.

1. Scope.

The real procedure DRUMCROUT 4 written in GIER-ALGOL III solves a system of linear equations $A \times y = b$, where A is a square matrix of order n, y is the solution and b the right hand side, both are vectors of dimension n.

The procedure can solve a system with several right hand sides, b, without repeating the triangularization of matrix A, i.e. it can solve a matrix equation $A \times Y = B$ where A is a square matrix of order n, Y and B are matrices with n rows and m columns.

DRUMCROUT 4 assumes that the matrix A and a right hand side b are stored on drum. The solution y will be stored on drum, by the procedure.

2. Method.

The solution is based on Crouts method with pivoting such as described for procedure CROUT in SA-10/1 sections 2 and will not be repeated here.

DRUMCROUT 4 is equivalent to DRUMCROUT 1 with respect to its action but most of the arithmetic work is done in machine-code. Special care has been taken in order to reduce the number of transfers of drum tracks holding the data.

The use of machine-code has resulted in considerably shorter computing time compared with DRUMCROUT 1.

3.Use of the procedure.

The procedure consists of two parts, the algol procedure and the machine-code which again is separated in two parts,the first being an initializing sequence, while the second holds the subroutines used in the computations. This splitting up of the machine-code is done in order to save place in the stack.

The determinant of the coefficient matrix is obtained as the value of DRUMCROUT 4 during the first call of the procedure, with repeat = false. In calls with repeat = true the value of DRUMCROUT 4 is undefined.

The procedure call must be of the form:

```
DRUMCROUT2 (n,ta,tb,ty,tf,tp,tr,s1,l1,s2,l2,repeat,exit);
```

where the parameters are:

n: integer, the order of coefficient matrix;

ta: integer, the value of drumplace corresponding to the storage on the drum of the first row of the coefficient matrix A of dimension [1:n,1:n]. This matrix is stored compactly row by row but so that each row starts at the beginning of a drum track. When stored in this way the values of drumplace for two elements in the same columnn but on consecutive rows will differ by a quantity which is divisible by 40(= number of words on a drum track). The coefficient matrix A must be stored on the drum before the first call of the procedure. During the first call of the procedure (repeat = false) the elements of A are changed because A is triangulated. The triangulated matrix must not be changed if subsequent calls with new right hand sides (repeat = true) are wanted.

tb: integer, the value of drumplace corresponding to the storage on the drum of a real array of dimension [1:n] which holds the right hand sides used in turn.

ty: integer, the value of drumplace corresponding to the storage on the drum of a real array of dimension [1:n] which holds the solution vector y. It may be remarked that the value of ty can be set equal to the value of tb, in this case the right hand side will be replaced by the solution.

tf: integer, the value of drumplace corresponding to the storage on the drum of a real array of dimension [1:n] which holds the factors computed and used in equilibration of the coefficient matrix during the first call (repeat = false) of the procedure. These factors must not be changed because they are used in all subsequent calls (repeat = true) with the same coefficient matrix but with new right hand sides.

tp: integer, the value of drumplace corresponding to the storage on the drum of an integer array of dimension [1:n]. This array holds the pivots originating from the triangularization of the coefficient matrix during the first call (repeat = false) of the procedure. The pivots must not be changed because they are used in all subsequent calls (repeat = true) with the same matrix , but with new right hand sides.

tr: integer, the value of drumplace corresponding to the storage on the drum of an integer array of dimension [1:n]. This array holds the values of drumplace corresponding to the storage of the rows of the coefficient matrix A and they are stored during the equilibration process in the first call (repeat = false of the procedure. These drumplace numbers are permutated during the triangularization according to the pivoting and must not be changed because they are used in all subsequent calls (repeat = true) of the procedure with the same matrix, but with new right hand sides.

s1,s2: integer, the value of the standard variable drumplace, when the first respectively the second part of the machinecode is stored on the drum by the standard procedure gierdrum (see below). They must not be changed throughout the program.

l1,l2: integer, the number of machine-words occupied by the first respectively the second part of the machinecode such as supplied by the standard procedure gierdrum (see below). They must not be changed throughout the program.

repeat: boolean. During the first call of DRUMCROUT 4 this parameter must have the value false. In all subsequent calls with the same coefficient matrix but with new right hand sides it must have the value true.

exit: a label to which jump is made from DRUMCROUT 4 if the coefficient matrix is singular.

Before the call of DRUMCROUT 4 the machine-code must be stored on the drum. Assuming that s1 has been assigned an appropriate value one writes in a suitable place in the program using the procedure, the four consecutive algolstatements.

drumplace:= s1; gierdrum (‹<initia›, l1);

s2:= drumplace; gierdrum (‹<subrou›, l2);

The procedure administers by itself the transfer of machinecode from the drum to the fast memory storage.

The first parameters ‹<initia›, respectively ‹<subrou› are code identifications for the first respectively the second part of the machine-code. They must be written exactly in this way. To the second parameter l1, respectively l2 are assigned the number of machine-words in the first respectively the second part of the code. This assignment is done by the standard procedure gierdrum.

4. Running time and storage requirements.

The speed of computations depends of course strongly on how the loop structure of the algol procedure matches with the tracks of the drum on which the procedure is stored.

The algol procedure requires 21 drumtracks, the first part of the machine-code 51, the second part 54 words in the fast memory storage.

Tests of the procedure with different systems have shown the following computation times:

order of system	computation time
20	32 sec
40	2 min 0 sec + 5 sec for every new right hand side
41	2 min 26 sec + 7 sec - - -
60	4 min 54 sec + 9 sec - - -
80	8 min 50 sec + 13 sec - - -
81	10 min 30 sec + 15 sec - - -

5. References.

1. Communications of the ACM, 4, 1961, pp 176-77

2. SA-10/1 CROUT ALGOL-procedure February 1967

3. SA-22/2 DRUMCROUT 1 - April 1967

Ole Lang Rasmussen.

6. Algorithm.

```
comment A.E.K. April the 30th 1967.-the procedure DRUMCROUT 4 written in GIER-
ALGOL III solves a system of linear equations Axy= b with different b as descri-
bed in SA-69/1;

real procedure DRUMCROUT4(n,ta,tb,ty,tf,tp,tr,s1,l1,s2,l2,repeat,exit);
value n,ta,tb,ty,tf,tp,tr,s1,l1,s2,l2;
integer n,ta,tb,ty,tf,tp,tr,s1,l1,s2,l2;
boolean repeat; label exit;
begin
integer i,j1,j2,j3,k,imax,gem,p,m,r,N;
real t,q,h,det,detfactor;real array a[1:n]; integer array row,pivot[1:n];
boolean  e1,e2,e3,e4,e5,e6;boolean  array code2[1:12];
gem:=drumplace;drumplace:=s2;fromdrum(code2);
N:=(n:40)×40; if N<n then N:=N+40;
begin boolean  array code1[1:11];
drumplace:=s1; fromdrum(code1);
gierproc(code1[4],code2,e1,e2,e3,e4,e5,e6,j1,j2,j3,n,t);
gierproc(code2[2],a);
if repeat then go to right hand side;
det:=detfactor:=1; k:=ta;
begin comment equilibration; array a1[1:N];
gierproc(code1[2],a1);
for i:=1 step 1 until n do
begin
drumplace:= row[i]:=k;fromdrum(a1); h:=gier(e1);
if h=0 then goto exit;
a[i]:=t:=2↑(-entier(ln(h)/0.693147181+1)); gier(e2);
detfactor:=detfactor×t; drumplace:=k;k:=k+todrum (a1);
end i;
drumplace:=tf; to drum(a);
end equilibration;
end code 1;
for p:=40 step 40 until N do
begin comment  triangularization;
m:=if p=40 then 0 else 1;
k:=p-40;
for k:=k+1 while k<n∧k≤p do
begin
t:=0;
begin array a1[m:p-40],a2[p-39:p];
gierproc(code2[4],a1); gierproc(code2[6],a2);
for i:=k step 1 until n do
begin
if p>40 then  begin drumplace:=row[i]-N+p-40;fromdrum(a1) end;
drumplace:=row[i]-N+p;fromdrum(a2);
if k≠1 then
begin
if k-1>p-39 then a2[k-1]:=a2[k-1]×q else a1[k-1]:=a1[k-1]×q;
j1:=1; j2:=k-1; j3:=p-40;
h:=a2[k]:=a2[k]-gier(e3);
drumplace:=row[i]-N+p;todrum(a2);
if k-1<p-39 then begin drumplace:=row[i]-N+p-40;to drum(a1)end
end k≠1 else h:=a2[1];
if abs(h)≥t then begin t:=abs(h); imax:=i end;
end i;
pivot[k]:=imax;
if imax≠k then
```

```
begin comment interchange drumplace numbers;
det:=-det;i:=row[k];row[k]:=row[imax];row[imax]:=i
end imax≠k;
end;
begin array a1[1:p];
gierproc(code2[4],a1);
drumplace:=row[k]-N+p; fromdrum (a1); t:=a1[k];
if t=0 then go to exit;det:=det×t;
if k<nthen
begin
r:=if k=p then p+1 else p-39;
begin array a2[k+1:n],a3[r:N];
gierproc(code2[6],a2); gierproc(code2[8],a3);
a:=1/t; j1:=k+1; j2:=n; gier(e4);comment set element of array equal to zero;
for i:=1 step 1 until k-1 do
begin
drumplace:= row[i]; fromdrum(a3);a[i]:=a3[k+1]; j3:=i; gier(e5);
end i;
drumplace:=row[k]; fromdrum (a3); gier(e6); a[k]:=a3[k+1];
drumplace:=row[k];to drum(a3);
end; end k<n;
end;
end k;
end p,triangularization;
drumplace:=tr;todrum(row);drumplace:=tp;todrum(pivot);
DRUMCROUT 4:=det/detfactor;
right hand side:
drumplace:=tb; fromdrum(a);
begin comment multiply right hand side with equilibration factors;
array f[1:n];
drumplace:=tf;fromdrum(f);
for i:=1 step 1 until n do a[i]:=a[i]×f[i];
end of multiplication;
drumplace:=tp; fromdrum(pivot);
drumplace:=tr; fromdrum(row);
for p:=40 step 40 until N do
begin comment elimination of right hand side;
array a1[1:p];
gierproc(code 2[4],a1); k:=p-40;
for k:=k+1 while k<n∧k≤p do
begin
t:=a[pivot[k]];a[pivot[k]]:=a[k]; a[k]:=t;
drumplace:= row[k]-N+p; fromdrum(a1);
j1:=1; j2:=k-1; j3:=k; a[k]:=a[k]-gier(e3);
end k;
end p, righthandside;
for p:=N-39 step -40 until 1 do
begin comment  backsubstitution;
array a1[p:N];
m:=if p=N-39 then nelse  p+39;
for k:=m step-1 until p do
begin
drumplace:= row[k]; fromdrum(a1);
j1:=k+1; j2:=n ; j3:=n+1;
a[k]:=(a[k]-gier(e3))/a1[k];
end k;
end backsubsitution;
drumplace:=ty;todrum(a); drumplace:=gem;
end DRUMCROUT4;
```

7. Machine code.

```
[machine code to DRUMCROUT 4]
b d7,e5
b b3,c5
;code1,initializing,gierproc(code1[4],code2,e1,e2,e3,e4,e5,e6,j1,j2,j3,n,t)
n
e1:c1: qq 10.3+499+57.15+19.21+57.27+37.33+57.39;¢<initia>,working cell
r
        hsb3
        gac4        ,hrs1       ; store basis-constant for array a1
        hsb3                    ; Radr:= basis-constant for code2
        arc2        ,grc1       ; jump order to address code2[1]-1
        arnb1 D
        arc2        ,gr(p5)     ; jump to greatest element in variable e1
        arnb2 D
        arc2        ,gr(p6)     ; jump to multiply with common factor in variable e2
        arn m d1 D
        arc1        ,gr(p7)     ; jump to subroutine 1 in variable e3
        arn m d2 D
        arc1        ,gr(p8)     ; jump to subroutine 2 in variable e4
        arn m d3 D
        arc1        ,gr(p9)     ; jump to subroutine 3 in variable e5
        arn m d4 D
        arc1        ,gr(p10)    ; jump to subroutine 4 in variable e6
        arn m d5 D              ; Radr:= reladr in code2 of order b2
        arc1        ,gac2       ; store abs.adr. of order b2 in code 2
        it(p11)     ,pa(c2)     ; store abs. adr[j1]
        arn m d6 D              ; Radr:=reladr.in code 2 of order 1b2
        arc1        ,gac2       ; store abs.adr.of order 1b2 in code2
        it(p12)     ,pt(c2)     ; store abs.adr[j2]
        arn m d7 D              ;Radr:=reladr. in code 2 of order b3
        arc1        ,gac2       ; store abs.adr. of order b3 in code 2
        it(p13)     ,pa(c2)     ;store abs. adr[j3]
        arfn(p14)   ,tkfm1      ; Radr:= n
        gac3        ,it(p15)    ; store n
        pac5        ,hrs1       ; store abs. adr[t],return to ALGOL.
c2:     hv                      ; hv-instruction, working cell
b3:     it(p4)      ,pa r+1     ;store address of dopevector
        arn m0 t-1              ;R:=lengthx2↑(-39)
        ar(r-1)t-1              ;R:=(length+constant)x2↑(-39)
        tkm30       ,grc1       ;store(length+constant)x2↑(-9)
        arn p4      ,tkm 10     ;Radr:=last element+1
        src1        ,hrs1       ;Radr:=basisadr-constant

b       a4
[greatest element]
b1:     gpa2        ,ppm1       ; store p-reg. p:=1
        grfn c1                 ; working cell:= 0
c3:a1:  it m0       ,bs p       ; if p>n then go to next order
a2:     ppm0        ,hva4       ; retabl.p-reg. jump
c4:     anfn p0     ,snfc1      ; RF:= abs(a1[p])-abs(c1)
        hv a3 LT                ; if RF<0 then go to a3
        anfn(c4)    ,grf c1     ; if RF≥0 then store abs(a1[p])
a3:     pp p+1      ,hva1       ; p:= p+1, jump
a4:     anfn c1     ,hrs1       ; RF:= max abs(a1[p]), return to ALGOL
e
```

```
b       a2
[multiply with common factor]
b2:     gp a2           ,pp m1          ; p:=1
a1:     it(c3)          ,bsp            ; if p>n then go to next order
a2:     pp m0           ,hrs1           ; retabl p-reg. retur to ALGOL
        arfn(c4)                        ; RF:=a1[p]
c5:     mkf m0          ,grf(c4)        ; RF:=a1[p]xt, store RF
e2:     pp p+1          ,hva1           ; p:= p+1, jump
e
e

b       b7,c8
[code2, subroutines 1,2,3,4]
n
e3:c1:qq 10.3+20.9+38.15+41.21+50.27+20.33+18.39;<<subrou>, workng cell
r
        hsb1
        gac5            ,hrs1           ; store basis-constant for array a
        hsb1
        gac7            ,hrs1           ;store basis-constant for array a1
        hsb1
        gac6            ,hrs1           ; store basis-constant for array a2
        hsb1
        gac8            ,hrs1           ; store basis-constant for array a3
b1:     it(p4)          ,pa r+1         ; store address of dopevector
        arn m0 t-1                      ; R:= lengthx2↑(-39)
        ar(r-1) t-1                     ; R:=(constant+length)x2↑(-39)
        tkm 30          ,grc1           ; (constant+length)x2↑-9)
        arn p4          ,tkm 10         ; Radr:=last element+1
        src1            ,hrs1           ; Radr:= basisadr-constant
[storing the values of j1,j2,j3, ≤ 511]
[d5]b2: arfn m0         ,tkfm1          ;Radr:=j
[d6]    gac2            ,arfn m0        ; store j1,Radr:=j2
        tkfm1           ,gac3           ; store j2
[d7]b3: arfn m0         ,tkfm1          ;Radr:=j3
        gac4            ,hrs1           ; store j3, return

b       a2
[subroutine 1, innerproduct 1]
[d1]b4: hsb2                            ; store j1,j2,j3,
        gpa1            ,grfnc1         ; store p-reg., working cell:=0
c2:     ppm0                            ; p:=p1
c3:a2:  it m0           ,bs p           ; if p>j2 then next order
a1:     ppm0            ,hrs1           ; retabl. p-reg. return to ALGOL
c5:     arfn p0                         ; RF:= a[p]
c4:     it m0           ,bsp            ; if p>j3 then next order
c6:     mkf p0 V                        ; RF:=a[p]xa2[p]
c7:     mkf p0                          ; RF:=a[p]xa1[p]
        arfc1           ,grfc1          ; accumulate product
        ppp+1           ,hva2           ; p:=p+1,jump
e
ba2
[subroutine 2, set zero to array a2]
[d2]b5: hsb2                            ; store j1 and j2,j3 not used in this routine
        gpa2            ,pp(c2)         ; store p-reg. p:=j1
a1:     it(c3)          ,bsp            ; if p>j2 then next order
a2:     ppm0            ,hrs1           ; retab p-reg. return to ALGOL
        grfn(c6)        ,ppp+1          ; a2[p]:=0
        hva1                            ; jump
e
```

```
b a2
[subroutine 3, innerproduct 2]
[d3]b6:  hsb3                          ; Radr:=j3
         gpa1          ,ppm0           ; store p-reg. p:=0
         ar(c7) D                      ; Radr:=absadr a1[j3]
         tkm-10        ,gta2           ; absadr a1[j3]
         pp(c2)                        ; p:=j1
a3:      it(c3)        ,bsp            ; if p>j2 then next order
a1:      ppm0          ,hrs1           ; retab. p.reg. return to ALGOL
c8:a2:   arfn p0       ,mkfm0          ; RF:= a3[p]xa1[j3]
         arf(c6)       ,grf(c6)        ; a2[p]:=a2[p]+a1[j3]xa3[p]
         ppp+1         ,hva3           ; p:=p+1,jump
e

b a2
[subroutine 4]
[d4]b7:  gpa2          ,pp(c2)         ; store p-reg. p:=j1
a1:      it(c3)        ,bsp            ; if p>j2 then next order
a2:      ppm0          ,hrs1           ; retab.p-reg.return to ALGOL
         arfn(c8)      ,srf(c6)        ; RF:=a3[p]-a2[p]
         grf(c8)       ,ppp+1          ; a3[p]:=RF
e4:      hva1                          ;jump
e
d1=b4-c1+1,d2=b5-c1+1,d3=b6-c1+1,d4=b7-c1+1,
d5=b2-c1+1,d6=d5+1,d7=d6+2
e

e5: hsm1
n
h kompud/3
3c
4c
qo e1.19+e2.39
hs 1
h kompud/3
3c
4c
qo e3.19+e4.39
zq 0
u e5
e
e
s
```

DANISH ATOMIC ENERGY COMMISSION
Research Establishment Risø
Reactor Physics Department
Computer Group.

September 1967
SA - 81/1
25 copies.

Chebyshev approximation

ALGOL-procedure CHEBAPPROX.

1. Scope

The procedure approximate a discrete function y[i]=f(x[i]) i=1,2 . . . m, defined on a closed interval x[1] ≤ x[i] ≤ x[n] by a set of m points (x[i],y[i]). The approximation is given as a finite serie of the form

$$y = f(x) = (a[0]/2) \times T0(u) + a[1] \times T1(u) + \ldots + a[n] \times Tn(u) \qquad (1)$$

with $u = (2 \times x - x[m] - x[1])/(x[m] - x[1])$, $-1 \leq u \leq 1$, $x[1] \leq x \leq x[m]$.

Tj(u) is the Chebyshev polynomials of order j, j = 0,1,2, . . .n, which are defined on the interval $-1 \leq u \leq 1$.

The procedure computes the coefficients a[0], a[1], ...a[n] and the approximation is characterized by the fact that the absolute value of maximum deviation of the approximation from the given point set (x[i],y[i]) is minimum.

2. Method

Survey of the theory.

The theoretical background for the procedure is the second algorithm of Remez. We seek a vector c such that the maximum norm on an interval [a,b] of the function

$$r(x) = f(x) - P(x) = f(x) - (c1 \times g1(x) + c2 \times g2(x) + \ldots + cn \times gn(x)) \text{ is minimum.}$$

The set of functions gi(x),i = 1,2, ...n, is assumed to satisfy the Haar condition:

1. gi(x),i= 1,2 ....n, are continuous on [a,b]
2. any set of n vectors of the form
   [g1(x), g2(x), ... gn(x)]

is independent. Expressed otherwise, each determinant.

$$D[x1,x2,..xn] = \begin{vmatrix} g1(x1) & \ldots\ldots & gn(x1) \\ \vdots & & \vdots \\ g1(xn) & \ldots\ldots & gn(xn) \end{vmatrix} \neq 0 \qquad (2)$$

for distinct x1,x2,.....xn.

We will show that all determinants D[x1,x2, ...xn] with x1 < x2 < ..... xn have the same sign.

The proof follows that given in (ref.[1]).

We assume the converse i.e. that there exists two sets

$$x1 < x2 < \ldots\ldots\ldots < xn, \text{ and } y1 < y2 < \ldots\ldots\ldots < yn$$

for which we have

$$D[x1, x2, \ldots\ldots\ldots xn] < 0 < D[y1, y2 \ldots\ldots\ldots\ldots yn]$$

Since D is continuous in the variables there exists a k, $(0 \leq k \leq 1)$ such that

$$D[k \times x1 + (1-k) \times y1, \ldots\ldots\ldots\ldots k \times xn + (1-k) \times yn] = 0$$

From the Haar condition is then follows that

$$k \times xi + (1-k) \times yi = k \times xj + (1-k) \times yj$$

for some distinct i and j. Hence xi - xj and yi - yj must have opposite signs which contradicts our assumption regarding xi and yi. This proves the proposition. According to the alternation theorem (ref[1]) the necessary and sufficient condition for that P(x) is the best approximation to f(x) on a given subset X in the interval [a,b] is, that r(x) exhibits at least n+1 alternations on X. i.e. $r(x[i])=-r(x[i-1]) = \pm||r||$ for i=1,2 ....n+1. x[i] is a nember of X and $||r|| = \max |r(x)|$.

Remez' algorithm proceeds as follows:

Starting with an arbitrary set X of n+1 elements (points) in the interval [a,b] we can compute a vector c for which max $|r(x[i]))|$ is minimum. From the alternating theorem follows that r(x[i]) are of equal magnitude $||r||$ but of alternating signs. Then we seek a new subset Y such that $|r(y[i])| \geq ||r||$ and of alternating sign. It should be remarked that r(y[i]), the deviations in the new subset, are not necessarily of the same magnitude. This completes the first computing cycle. In the next cycle we use the subset Y in place of X.

The successively generated P(x) can be shown to converge uniformly to the best approximation (ref[1]).

. . . . . .

Application.

We now apply the theory on the discrete problem. As our g - functions we take the Chebyshev polynomials Tj(x) and the elements of the c vector is denoted by a[0]/2, a[1], ...a[n].

The computation of the coefficient a[i], i = 0,1,2 ...n is done by repeated solution of a linear system of order n + 2 with n + 2 variables.

The equation systen to be solved is of the form:

$$\begin{aligned} &a0/2 \times T0(x[1]) + a1 \times T1(x[1]) + \ldots\ldots + an \times Tn(x[1]) + d = y[1] \\ &a0/2 \times T0(x[2]) + a1 \times T1(x[2]) + \ldots\ldots + an \times Tn(x[2]) - d = y[2] \end{aligned} \qquad (3)$$

. . . . . .

. . . . . .

$$a0/2 \times T0(x[n+2]) + a1 \times T1(x[n+2]) + \ldots\ldots + an \times Tn(x[n+2]) - (-1)^{(n+2)} d = y[n+2]$$

where

$$x[1] < x[2] < \ldots\ldots\ldots < x[i] < x[i+1] < \ldots\ldots < x[n+2] \qquad (4)$$

are the reference points. The n+2 unknowns to be found are a0/2, a1, an and the deviation d in the reference points.

It can easily be verified that the system of Chebyshev polynomials [T0(x), T1(x), ...Tn(x)] satisfy the Haar condition and that the determinant

$$\begin{vmatrix} T0(x[1]), & T1(x[1]), & T2(x[1]), & \ldots\ldots\ldots\ldots & Tn(x[1]) & +1 \\ To(x[2]), & T1(x[2]), & T2(x[2]), & \ldots\ldots\ldots\ldots & Tn(x[2]) & -1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ To(x[n+2]), & T1(x(n+2]), & T2(x[n+2]), & \ldots\ldots\ldots\ldots & Tn(x[n+2]) & -(-1)\uparrow(n+2) \end{vmatrix} \neq 0. \quad (5)$$

Since $x \uparrow j$, $0 \leq j \leq n$ can be written as a linear combination of $Ti(x)$, $0 \leq i \leq j$, it is easily verified that the determinant in (5) can be transformed to

$$D = \begin{vmatrix} 1, & x[1], & x[1]\uparrow 2, & & x[1]\uparrow n, & +1 \\ 1, & x[2], & x[2]\uparrow 2, & & x[2]\uparrow n, & -1 \\ & & & & & \\ 1, & x[n+2], & [n+2]\uparrow 2, & & x[n+2]\uparrow n, & -(-1)\uparrow(n+2) \end{vmatrix} \quad (6)$$

We observe that any of the n+1 minors of order n+1 which is a cofactor to the elements in the last columm is a Wandermonde - determinant which is $\neq 0$ because of (4). Every subset of n+1 different row vectors of dimension n+1 (1, x[i], x[i]↑2, ..., x[i]↑n), i= 1,2 ... n+2, thus satisfy the Haar condition, and as shown above these minors all have the same sign.

Let M be the minor which is cofactor to element i in the last column we find for the determinant D in (6)

$$D = (+1) \times (M1+M2 + \ldots\ldots\ldots + Mn+2) \neq 0$$

Hence also the determinant in (5) is $\neq 0$

Starting with a set of n+2 approximately equally spaced reference points the procedure computes a set of coefficients a[i], i.... 0,1, ... n.

Then the deviations from the approximation are computed for all m points. In the reference points the deviation d has alternating sign but same magnitude. Among these m point the procedure selects those n+2 points in which the deviation with alternating signs are numerically greatest. Some of these may be the old reference points. These points work as new reference points.

In every computing cycle a set of values a[0], a[1], ... a[n], d is computed. The absolute value of d (the reference point deviation) is increased from one cycle to the next. At the same time the absolute value of the maximum deviation in the m points is decreased.

The iteration stops when the absolute value of d equals the absolute value of the maximum deviation or when the absolute value of d reach its maximum.

3. Use of the procedure.

The procedure call is of the type.

CHEBAPPPROX(n,m,x,y,a maxdeviation);

where

n:	integer is the order of approximation.
m:	integer is the number of points (x,y)
x:	real array x[1:m] contains the abseissae, and
y:	real array y[1:m] contains the ordinates of the m points
a:	real array a[0:n] contains the values of the coefficients in the Chebyshev approximation computed and stored by the procedure.
maxdeviation:	real contains on exit the maximum deviation in the m points.

The n and m must satisfy the inequality $m \geq n+2$.

The points (x,y) must be stored with increasing values of x, i.e. x[1] < x[2] < ..... < x[m-1] < x[m]:

In order to test that these conditions are satisfied the programmer can insert a small testprogram immidiately before the call of the procedure. Such a testprogram can looks as follows

```
if m < n+2 then
begin
outtext (‹<error:number of points too small or degree of approximation too large›);
goto end of program;
end;
begin integer i; boolean error; error:= false
for i:= 1 step 1 until m-1 do
if x[i] ≥ x[i+1] then
begin error:= true;
outtext (‹<
error: x[›); output (‹nd›,i); outtext (‹<] ≥x[›); output (‹nd›,i+1);
outtext (‹<]›);
end;
if error then goto end of program;
end;
```

When the coefficient a[0], a[i], ...a[n] has been found we can compute corresponding values of x and y by (1). This am be done by SA-96.

4. Reference

[1] E W. Cheney: Introduction to Approximation Theory. Mc. Graw Hill 1966.

O. Lang Rasmussen.

5. Algorithm.

```
procedure CHEBAPPROX(n,m,x,y,a,maxdeviation);
value n,m; integer n,m;real maxdeviation;real array x,y,a;
begin comment A.E.K. - September 1967.This procedure fits as described
              in SA 81/1 a discrete function given as m points (x,y)
              by a Chebyshevapproximation of order n:
              y=a0/2+a1×T1(x)+a2×T2(x)+..... +an×Tn(x)
              where Tn(x) is the Chebyshev-polynomial of degree n and
              an the corresponding Chebyshev coefficient. This fit by
              Chebyshev polynomials is the best approximation in the
              Chebyshev sense i.e. in the Chebyshev-norm;
real p,q,d,maxd; integer array IN[0:n+3];array b[1:n+2];
begin comment starting reference points;
integer i;
q := (m-1)/(n+1);
IN[n+] := IN[n+2] := m; IN[0] := 1;
for i := n+1 step -1 until 1 do IN[i]:=(i-1)×q+1;
p:=(x[m]+x[1])×0.5;q:=(x[m]-x[1])×0.5;
end starting reference points;
maxd:=0;
start:
begin comment computation of the Chebyshev coefficient and the deviation
              in the reference points by solving a linear system of n+2
              equations with n+2 variables;
real array A[1:n+2,1:n+2];
begin comment compute coefficient matrix and the right hand side;
real c,u0,u1,u2; integer i,j,k;
d:=1;
for i :=1 step 1 until n+2 do
begin
k:=IN[i]; c := 2×(x[k]-p)/q; A[i,1]:=1;u1:=0;u2:=-1;
for j:=2 step 1 until n+1 do
begin u0 :=c×u1-u2; u2:=u1; u1:=u0;A[i,j]:=c×u1×0.5-u2;end j;
A[i,n+2]:=d;d:=-d;b[i]:=y[k];
end i;
end coefficient matrix and right hand side;
begin comment solution of the linear system;
integer i,j,k,imax;real h,t;
for k:=1 step 1 until n+2 do
begin comment triangularization starts;
t :=0;
for i := k step 1 until n+2 do
begin
h := A[i,k];
for j:= 1 step 1 until k-1 do h := h-A[i,j] × A[j,k];
A[i,k] := h;
if abs(h)>t then begin t := abs(h); imax := i end;
end i;
if imax ≠ k then
begin
for i:=1 step 1 until n+2 do
begin comment interchange rows k and imax;
t := A[k,i]; A[k,i] := A[imax,i]; A[imax,i] := t;
end interchange of rows;
t:=b[k];b[k]:=b[imax];b[imax]:=t;
endimax≠k;
```

```
h := 1.0/A[k,k];
for i:=k+1 step 1 until n+2 do A[i,k]:= h×A[i,k];
for i:=k+1 step 1 until n+2 do
begin
h := A[k,i];
for j:=1 step 1 until k-1 do h := h-A[k,j] × A[j,i];
A[k,i] := h;
end i;
h:=b[k];for j:=1 step 1 until k-1 do h:=h-A[k,j]×b[j];b[k]:=h;
end triangularization;
for k :=n+2 step -1 until 1 do
begin comment backsubstitution for Chebyshev coefficients and the deviation d;
h :=b[k];
for j:=k+1 step 1 until n+2 do h := h-A[k,j] ×b[j];
b[k]:=h/A[k,k];
end k;
b[1]:=2×b[1]; d:=b[n+2];
end solution of linear system;
end chebyshev coefficients;
begin comment compute deviations and new reference points;
real array T[1:m]; integer imax;real Tmax;
begin comment deviations; real c,u0,u1,u2; integer i,j,k,s;
Tmax:=0; s:=1;k:=IN[s];
for i := 1 step 1 until m do
begin
if i=k∧s<n+2 then begin T[i]:=d;d:=-d;s:=s+1;k:=IN[s] end else
begin
u2 := u1 := 0; c := 2 ×(x[i]-p)/q;
for j := n+1 step -1 until 2 do
begin
u0 :=b[j] + c × u1 - u2; u2 := u1; u1 := u0
end j;
T[i]:=y[i]-((b[1]+c×u1)×0.5-u2);
end else-statement;
if Tmax< abs(T[i]) then begin Tmax := abs(T[i]); imax := i end;
end i;
if Tmax=abs(d) then
begin maxdeviation:=Tmax;for i:=0 step 1 until n do a[i]:=b[i+1];
goto END end;
if d≠0 then
begin if abs(d)>maxd then
begin maxd:=abs(d);maxdeviation:=Tmax;
for i:=0 step 1 until n do a[i]:=b[i+1];
end else goto END end;
end deviations;
begin comment new reference points;
integer i,j,k,imin,jmax;real Ti,Tj,extremum;booleanfound;
j:=0;found:=false;
for j:=j+1 while j<n+2∧-,found do
if IN[j-1]<imax∧imax<IN[j+1] then
begin comment an interval containing T[imax] is found;
if T[IN[j]]×T[imax]>0 then begin found:=true;jmax:=j end else
begin comment T[IN[j]] and T[imax] are of opposite sign;
if imax<IN[1] then
begin comment move the reference point set to right;
for i:=n+2 step -1 until 2 do IN[i]:=IN[i-1];found:=true;jmax:=j;
end else if imax>IN[n+2] then
begin comment move the reference point set to left;
for i:=1 step 1 until n+1 do IN[i]:=IN[i+1];found:=true;jmax:=j;
end;end;end;
```

```
IN[jmax]:=imax;
if d≠0 then
for j:=jmax+1 step 1 until n+2,jmax-1 step -1 until 1 do
begin comment after having placed a reference point in the point of
              maximum deviation we proceed by changing the position of
              the remaining reference points;
k:=IN[j];Tj:=T[k];extremum:=abs(Tj);imin:=IN[j-1];imax:=IN[j+1];
for i:=imin step 1 until imax do
begin comment the deviations T[imin] and T[imax] being of the same sign,
              we look for the maximum deviation of opposite sign,T[i],the
              point of which is the newreference point IN[j];
Ti:=T[i];
if TixTj>0 then
begin if abs(Ti)≥extremum then begin extremum:=abs(Ti);k:=i end end;
end i;
IN[j]:=k;
end j;
end new reference points;
end deviations and new reference points;
goto start;
END:end CHEBAPPROX;

end of program;
```

DANISH ATOMIC ENERGY COMMISSION
Research Establishment Risø
Reactor Physics Department
Computer Group

November 9th 1965
SA - 85
50 copies

Determination of a real zero of a real function
ALGOL-procedure: HYPAR.

1. Scope.

The following boolean procedure determines with prescribed accuracy a zero of a real function in a prescribed real interval. If the values of the function at the end points of the interval differ from zero and have the same sign, the value of HYPAR will be true, otherwise it will be false.

2. Method.

The zero is found iteratively: The function is approximated by a hyperbola with a vertical asymptote. This gives a four point iteration scheme, which is used, if the four previous points are on the same branch of the hyperbola. Otherwise parabolic interpolation is used. If the actual convergence rate twice in succession is lower than that corresponding to bisection, this latter method is used.

The two extra points needed to start the process are generated by means of bisection and parabolic interpolation.

The iterations are stopped, when the horizontal distance from the last point to a point on the other side of the x-axis is less than the input parameter eps.

3. Use of the procedure.

The procedure will be copied into the program where the following comment is written:

```
comment library HYPAR;
```

The procedure call must have the form:

```
HYPAR (F,x,y,d,eps);
```

F is the name of a real procedure (with one formal parameter) determining the function for which a zero is wanted.

x is the name of the zero (a real variable). The value of x must equal a first approximation to the zero.

y is the name of a real variable, whose initial value must be F(x).

d is the name of a real variable, whose initial value is determined in the following way:

abs(d) is the accuracy with which x is thought to be known, so that simple functioning of the procedure is obtained, when a zero is situated in one of the intervals $(x, x \pm d)$.

sign(d) is determined so that d has the same sign as the function values immediately to the left of the zero.

eps is the (real) accuracy wanted for x; a relative accuracy rel way be prescribed by inserting rel × (name of zero) in the place of eps in the procedure call.

It is suggested that one should use the first part of HYPAR for finding an interval containing a zero in case one has no a priori knowledge of such an interval.

For instance one might use the following statements to find a smallest zero z to the right of some point x1, when it is known that the distance between z and the next zero is greater than a certain function (assumed, for simplicity, nondecreasing) $f(z)>0$, and that $z<x2$:

```
x:=x1; y:=F(x1); s:=sign(y);
for d:=f(x)×s while HYPAR (F,x,y,d,rel×x) do
begin if x>x2 then goto Abrahams skød end;
```

The idea of using some variables (x,y and d) as both input and output parameters is due to Erling Jensen.

4 Reference.

For a further discussion of this and related methods see Regnemaskinememo No 11.

5. Algorithm.

```
boolean procedure HYPAR(F,x,y,d,eps);
real x,y,d,eps;
real procedure F;

begin
real x1,x2,x3,x4,y1,y2,y3,y4,A,B,C;
integer p;
HYPAR:=false;
x1:=x; y1:=y;
if y=0 then goto out;
x2:=x:=x+d×sign(y); y2:=y:=F(x);
if y=0 then goto out;
if y1×y2>0 then begin HYPAR:=true; goto out end;
x :=(x1+x2)/2;
y := F(x);
if y = 0 then goto out;
x3:=x2; y3:=y2;
if sign(y) = sign(y2) then
begin x3 :=x1; x1:=x2; y3:=y1; y1:=y2; end;
x2:=x; y2 := y;
d := abs(x3-x2);
p:=0;
goto par;

hypar:
A := (y1-y4)×(y2-y3)×(x1-x2)×(x4-x3)+(x3-x2)×(x1-x4)×(y1-y2)×(y4-y3);
B := (y1×(x3-x2)-y3×(x1-x2))×(y4-y2)×(x1-x2)×(x3-x2)
+(y4×(x3-x2)-y3×(x4-x2))×(y2-y1)×(x3-x2)×(x4-x2)
+(y1×(x4-x2)-y4×(x1-x2))×(y2-y3)×(x1-x2)×(x4-x2);
C:=(x1-x2)×(x3-x2)×(x4-x2)×y2×((x4-x3)×(y1-y3)+(x3-x1)×(y4-y3));

if sign((x1-x2)×(y3-y2)+(x3-x2)×(y2-y1))
≠(if (x4-x2)×(x3-x2)<0 then sign((x1-x2)×(y4-y2)+(x4-x2)×(y2-y1))
else sign((x4-x2)×(y3-y2)+(x3-x2)×(y2-y4)))
then
```

```
par:
begin
x4 :=x2;
A := (y2-y1)×(x3-x2)+(y3-y2)×(x1-x2);
B:=(y2-y1)×(x3-x2)↑2+(y3-y2)×(x1-x2)↑2;
C:=y2×(x1-x2)×(x3-x2)×(x3-x1);
end;
y:=B↑2-4×A×C;
if y≤0 then y:=0 else y:=2×C/(B+sign((x3-x2)×C)×sqrt(y));
if abs(y)>d/2 then
begin if p=1 then y:=(x3-x2)/2; p:=1-p; end else p:=0;
if y×(x3-x2-y)<0 then y:=(x3-x2)/2; x:=x+y;
d := abs(x2-x);
x4:=x1; y4:=y1;
y := F(x);
if y = 0 then goto out;
if sign(y) = sign(y3) then
begin
x1 := x3; x3 := x2;
y1 := y3; y3 := y2;
end
else
begin
x1 := x2; y1 := y2;
end;
x2:=x; y2 := y;
if abs(x3-x2) < eps then goto out;
if d>eps then  goto hypar;
if F(x+eps×sign(x3-x2) × y≤ 0 then goto out;
d :=( x3-x2)/2 × sign(y);
for x :=( x+x3)/2,x+d×sign(y) while abs(d) > eps do
begin d := d/2; y:=F(x); end;

out:
end of HYPAR ;
```

G. K. Kristiansen.

DANISH ATOMIC ENERGY COMMISSION
Research Establishment Risø
Reactor Physics Department
Computer Group

December 20th 1965
SA- 86
25 copies

Solution of algebraic equations

ALGOL - procedure LEHMERNEWTON.

1 Scope.

The procedure LEHMERNEWTON solves an algebraic equation of order n with complex coefficients

$$P(z) = a[n]\times z\uparrow n + a[n-1]\times z\uparrow(n-1) + \ldots\ldots\ldots + a[1]\times z + a[0] = 0$$

where $z = x + iy$.

2 Method.

The procedure is a combination of the procedure LEHMER([3],[4]) and the fast iterative Newton process. By means of the Lehmer procedure a circle is found, which contains only one root in its interior. Then the root is computed by Newton iteration.

The theoretical backgrund of the method is the lemmas of Cauchy and Rouche which is stated below without proofs as they can be found in every standard textbook on complex analysis.

Lemma 1. Let C be the unit circle, then

$$\int_C (1/(z-a))dz = 2\,i \quad \text{if } |a| < 1$$

$$\int_C (1/(z-a))dz = 0 \quad \text{if } |a| > 1$$

where the integral is taken along the unit circle in the positive sense.

Lemma 2. Let P(z) be a polynomial with no roots on the unit circle C. The number of roots of P(z) inside C is then given by

$$(1/2\pi i)\oint_C (P1(z)/P(2))dz$$

where P1(z) is the derivative of P(z). Multiple roots are counted according to their multiplicities.

Lemma 3. Let P(z) and Q(z) be two polynomials such that

$$|P(z)| < |Q(z)| \quad \text{for } |z| = 1$$

then Q(z) and P(z) + Q(z) have the same number of roots inside the unit circle (Rouche s theorem).

By means of these lemmas D.H. Lehmer has developed a powerful method for finding the roots of a polynomial. The description of the method is a little complicated and the reader is recommended to study the original work of D.H. Lehmer [1]. For a short introduction to the method the reader may consult [2]. In what follows an outline of the method will be given in order to clarify the computational process.

The circle

$$|z-c| = r$$

can by a linear transformation

$$z = r \times z1 + c$$

be transformed into the unit circle. If a polynomial f(z) has a root, a, inside the circle $|z-c| < r$ then the polynomial

$$g(z1) = f(rz1 + c)$$

has the root

$$b = (a - c)/r$$

where $|b| < 1$, i.e. inside the unit circle, then our problem to find a root inside a given circle can be replaced by the problem of finding a root inside the unit circle.

Given the polynomial

$$g(z) = a[n] \times z\wedge n + a[n-1] \times z\wedge(n-1) + \ldots\ldots\ldots + a[1] \times z + a[0]$$

where we assume $g(0) \neq 0$ (otherwise g(z) has the root z = 0). We form the polynomial

$$G(z) = z\wedge n \times \underline{g}(1/z) = \underline{a}[0] \times z\wedge n + \underline{a}[1] \times z\wedge(n-1) + \ldots\ldots + \underline{a}[n-1] \times z + \underline{a}[n]$$

where the coefficients $\underline{a}[0]$, $\underline{a}[1]$,....,$\underline{a}[n]$ are the complex conjugates of those of g(z) in reversed order. The polynomials T(g(z)) given by

$$T(g(z)) = \underline{a}[0] \times g(z) - a[n] \times G(z)$$

is certainly of lower degree than g(z) because the coefficient of $z\wedge n$ is zero. It is easily verified that the constant

$$\underline{a}[0] \times a[0] - \underline{a}[n] \times a[n]$$

is real. If $T(g(0)) \neq 0$ we can form a new polynomial

$$T(T(g(z))) = T2(g(z))$$

from T(g(z)) in the same way as T(g(z)) is formed from g(z). T2(g(z)) again is of lower degree than T(g(z)). Continuing in this way we obtain a finite sequence of polynomials

$$T(g(z)),\ T2(g(z)),\ T3(g(z)),\ \ldots\ldots\ ,\ Tk(g(z)) \qquad (1)$$

where $1 \leq k \leq n$ and

$$Tk(g(0)) = 0$$

We will denote by di the degree of Ti(g(z)) so that

$$n = d0 > d1 > d2 > \ldots\ldots\ldots > dk \geq 0$$

Lehmer has proved the following lemma

Lemma 4. Let $g(z)$ be a polynomial of degree d with no root on the unit circle C and m roots inside C. Let $T(g(0)) \neq 0$. Then $T(g(z))$ has no root on C and has m or d - m roots inside C according as $T(g(0))$ is positive or negative.

Based on these four lemmas Lehmer has proved the following

THEOREM.

Assume that $g(0) \neq 0$. If for some $h > 0$, $Th(g(0)) < 0$ then $g(z)$ has at least one root inside the unit circle. Otherwise if $Ti(g(0)) > 0$ for $1 \leq i < k$ and $T(k-1)(g(z))$ is constant, then $g(z)$ has no root inside the unit circle.

The detailed proof of this theorem is found in [1]. It shall be emphasized that the theorem says nothing about the existence of roots inside the unit circle in the case that the constant therm in one of the polynomials in (1) is zero.

Assuming that $g(z)$ of degree d0 has m roots inside and no roots on the unit circle (this can always be arranged by an appropriate chose of the circle which shall be transformed to the unit circle). Assuming that there is a minimum h such that

$$\begin{aligned} &Th(g(0)) < 0 \\ &Ti(g(0)) > 0 \quad \text{for } 0 \leq i < h \end{aligned} \tag{2}$$

Then applying lemma 4 in turn to the polynomials

$$g(z),\ T(g(z)),\ T2(g(z)),\ \ldots\ldots,\ T(h-1)(g(z))$$

which are computed such that their degrees form an arithmetic progression with the difference -1 starting with d0, we conclude that each of these polynomials has m roots inside C. Applying the lemma once more to $T(h-1)(g(z))$ we observe that $Th(g(z))$ has $d(h-1) - m$ roots inside C. Since dh is the number of roots of $Th(g(z))$ we have

$$dh \geq d(h-1) - m$$

or

$$m \geq d(h-1) - dh > 0$$

i.e. $m \geq 1$.

Hence $g(z)$ has at least one root inside C. If now $dh = 0$, i.e. $Th(g(z))$ is a constant which according to (2) is smaller than zero then $T(h-1)(g(z))$ is a polynomial of degree 1 and has only one root which is inside C. From lemma 4 it follows that also $g(z)$ has only one root inside C.

The procedure makes a systematical search for roots in the complex plan starting with the unit circle around origo, such as described in [1], [3]. Having thus found a circle which contains only one root this is finally computed by Newton iteration. During this iteration it is tested whether the iterands all stay within the circle found, otherwise the circle is decreased by repeating the Lehmer procedure. When a root is found it is refined by a Newton iteration in the original equation before it is removed from the equation. The procedure then continues the search for the next root. The roots are found roughly in order of increasing modulus.

3. Test of the procedure.

The procedure has been tested on several polynomial equations both with real and complex coefficients. The use of Newton iterations in connection with Lehmer s method has resulted in a faster procedure than the pure Lehmer procedure. It has been observed that the speeds are increased with a factor up to 6.

4. Use of algorithm.

The procedure call must be of the type

LEHMERNEWTON(n, A, tol);

where

n : declared as integer is the degree of the polynomial P(z)

A : declared as real array A[0:n,0:1] contains the complex coefficients. The real parts of the coefficients are stored in A[0,0],A[1,0]..... A[n,0] and the imaginary parts in A[0,1],A[1,1],.....A[n,1]. The coefficient a[0] is stored in A[0,0],A[0,1],a[1] in A[1,0],A[1,1],...., a[n] in A[n,0],A[n,1].
The roots found by the procedure are stored in A such that the first root is stored in A[n,0],A[n,1], the second in A[n-1,0],A[n-1,1],.... the nth root in A[1,0],A[1,1]. The real part of a root is stored in an element of A with second index 0, the imaginary part in an element of A with second index 1.

tol: declared as real is a measure of the accuracy of a root. When the ith approzimation z[i] to a root has been found and the(i+1)th approximation z[i+1] is known to be found within a circle with z[i] as centrum and tol as radius then z[i] is accepted as a root. If tol is chosen too small the procedure will come to an end when it is looking in vain for a root.

5. Reference.

[1] D.H.Lehmer, A machine method for solving polynomial equations, Journal of the Association for Computing Machinery vol 8,pp 151-162. (1962).

[2] Selected Numerical Methods, chapter IV,pp 278-280 publ. by Regnecentralen, Copenhagen 1962.

[3] Ole Lang Rasmussen: Solution of polynomial equations by the method of D.H. Lehmer, BIT 4, pp 250 - 60. (1965).

[4] SA-34, LEHMER, February 26th 1964.

6. Algorithm.

```
procedure LEHMERNEWTON(n,A,tol);
value n,tol;integer n;real array A;real tol;
begin comment A.E.K. december 22th 1965 -the procedure LEHMERNEWTON finds
approximate values for the roots to a polynomial equation P(z)=a[n]×z↑n+
a[n-1]×z↑(n-1)+...+a[1]×z+a[0]=0 with complex coefficients as described in
SA-86.The procedure is a combination of two procedures,the Lehmer-procedure
for an effective localization of a root and the fast Newton iteration process.
The Lehmer procedure is developed in accordance with the method devised by
D.H.Lehmer described in Journal of the A.C.M. volume (1961) pp 151-162.
The parameters are:
n:    degree of the polynomial equation.
A:    array of dimension A[0:n,0:1] containing the complex coefficients a[n],
      a[n-1],.....a[1],a[0] which are stored according to the following rules.
      Re(a[k]) in A[k,0]
      Im(a[k]) in A[k,1] for n > k > 0
      The roots z[1],z[2],....z[n] will be stored in A according to the same
      rules and such that the first root found will be stored in A[n,0],A[n,1],
      the second in A[n-1,0],A[n-1,1],etc.....the last in A[1,0],A[1,1].
tol:  parameter giving the tolerance of the solution. If z is some approxima-
      tion to a solution then this approximation is accepted if the next ap-
      proximation shall be found within a circle with center z and radius r
      where abs(z)×tol>r;
integer m,i,k,p,q;
real S,xc,yc,r,r1,r2,a1,a2,a3,newxc,newyc,temp;
real array a,b[0:n,0:1];
boolean B1,B2;

procedure SYNDIV(m,b,x,y,r,B);
value m,x,y,r; integer m;real x,y,r;real array b;boolean B;
begin comment The procedure SYNDIV is used with B having the value false when
a root to the polynomial equation P(z) = 0 has been found and shall be removed
from the equation. When B has the value true then SYNDIV makes a linear trans-
formation of a polynomial f(z) within a circle with center (x+iy) and radius r
to a polynomial g(z) within the unit circle;
integer i,j,k; real q,s,t,u,v;
q := 1;
if B then k:=m else k:=0;
for i:=0 step 1 until k do
begin s:=t:=0;
     for j:=m step -1 until i do
     begin u:=b[j,0]+x×s - y×t; v:=b[j,1]+x×t + y×s;
           b[j,0]:=s:=u; b[j,1]:=t:=v;
     end j;
     if B then begin b[i,0]:=u×q;b[i,1]:=v×q;q:=q×r end;
end i;
if ¬B then for j:=0 step 1 until m-1 do
begin b[j,0]:=b[j+1,0];b[j,1]:=b[j+1,1]end;
end SYNDIV;

procedure T(m,c,d);
value m; integer m; real array c,d;
begin integer j;
for j := 0 step 1 until m do begin c[j,0]:=d[j,0]; c[j,1] := d[j,1] end;
end T;
```

```
integer procedure ROOT(m);
value m;integer m;
begin comment The integer procedure ROOT(m) constitute the central part of
the algorithm.If the constant term in the polynomial g(z) is zero then g(z)
has the root zero and ROOT will be assigned the value 1.If the constant term
is different from zero then the procedure computes the coefficients in the
finite sequence of polynomials T(g(z)), T(T(g(z))),.... in turn and by examin-
ing the signs of the constant term in each of these polynomials answers the
fundamental question whether the polynomial equation g(z) = 0 has a root inside
the unit circle.If the answer is affirmative ROOT will be assigned the value 2
if there is only one root and the value 3 if there are more than one root in-
side the unit circle. If the answer is negative ROOT will be assigned the va-
lue -1.In the case that the constant term in one of the polynomials is zero
Lehmers theorem gives no answer to the question and ROOT will be assignedthe
value 0;
integer q,i,imax; real a1,a2,a3,a4,b1,b2,b3,b4;
if a[0,0]≠0∨a[0,1]≠0 then
begin
for q:=m step -1 until 1 do
begin
a1:=a2:=abs(a[0,0]) + abs(a[0,1]);
for i:=1 step 1 until q do
begin comment The coefficients for the polynomials computed by ROOT(m) are
apt to become either very large or very small in absolute value, so that they
can have values beyond the range of numbers for the computer used. In order to
refute this eventuality the coefficients are divided by a common factor. Let a
norm of a complex number be defined by ||z||=||x+iy||= |x|+ |y|. Find for the
actual polynomial the coefficients with the greatest and smallest norms and
find the greatest power of 2↑p (p integer) lesser than or equal to the square
root of their product. This power of 2 is the factor used in this context;
a3:=abs(a[i,0]) + abs(a[i,1]);
if 0<a3∧a3<a1 then a1:=a3;
if a3>a2 then a2:=a3;
end;
a3:= 2↑(-entier(ln(sqrt(a1)×sqrt(a2))/0.693147181));
for i :=0 step 1 until q do
begin a[i,0]:=a[i,0]×a3;a[i,1]:=a[i,1]×a3 end;
comment Compute the coefficients in the sequence of polynomials T(g(z)),
T(T(g(z))),.....;
imax:= q:2; a1:=a[0,0];a2:=a[0,1];a3:=a[q,0];a4:=a[q,1];
a[0,0]:=a1↑2+a2↑2-a3↑2-a4↑2;
a[0,1]:=a[q,0]:=a[q,1]:=0;
if a[0,0]>0 then
begin
for i:=1 step 1 until imax do
begin b1:=a[i,0]; b2:=a[i,1]; b3:=a[q-i,0]; b4:=a[q-i,1];
      a[i,0]:= a1×b1+a2×b2 - a3×b3 - a4×b4;
      a[i,1]:= a1×b2 - a2×b1 + a3×b4 - a4×b3;
      if q≠2×i then
      begin
      a[q-i,0]:= a1×b3 -a3×b1 + a2×b4 - a4×b2;
      a[q-i,1]:=a1×b4 -a4×b1 + a3×b2 -a2×b3
      end;
end i;
ROOT:=-1
end else
begin if a[0,0]<0 then begin if q=1 then ROOT:=2 else ROOT:=3 end
else ROOT :=0; goto exit end;
end q;
```

```
end else ROOT:=1;
exit:end procedure ROOT;

procedure NEWTON(m,r,xc,yc,a,b,exit);
value m,r;integer m;real r,xc,yc;real array a,b; label exit;
begin comment an approximate root is refined by a Newton process;
real a00,a01,a10,a11,s,x,y,newx,newy,delta1,delta2;
boolean firstiter;
x := xc; y := yc; firstiter:=true;
rep:
T(m,a,b); SYNDIV(m,a,x,y,1,true);
a00 := a[0,0]; a01 := a[0,1]; a10 := a[1,0]; a11 := a[1,1];
s := a10↑2+a11↑2;
newx := x-(a00×a10+a01×a11)/s;
newy := y-(a01×a10-a00×a11)/s;
if sqrt ((newx-xc)↑2+(newy-yc)↑2) > r then goto exit;
delta2:=sqrt((newx-x)↑2+(newy-y)↑2);
if firstiter then begin delta1:=delta2;firstiter:=false;goto label1 end;
if delta2<delta1 then delta1:=delta2 else goto label2;
label1:x := newx; y := newy; goto rep;
label2:xc := newx; yc := newy;
end NEWTON;

S :=1; a3:= 0.707106781;
T(n,b,A);
for m := n step -1 until 1 do
begin comment Beginning with a circle with center in the origin and radius S
the procedure starts the search for a root to the given polynomial P(z). For
the first root, S = 1, corresponding to the unit circle.The search for the
following roots starts with the greatest rootfree circle around origin found
during the computation;
r := S; xc := yc := 0;
comment Isolate a root by Lehmers procedure.Find radius r1 such that a poly-
nomial f(z) has a root inside an annulus bounded by concentric circles with
radius r1 and 2×r1 and the center (xc+iyc);
Lehmer:nextapprox:
i:=1;
L1:
B1:=B2:=false;
L2:
T(m,a,b); SYNDIV(m,a,xc,yc,r,true); q:= ROOT(m);
if q=1 then goto deflation;
if q>1 then
begin B1:=true;if B2 then goto L3;r:=r/2;goto L2 end;
if q=0 then
begin comment No answer to the question whether a root exists can be given,
repeat the process with a greater circle;
i:=i+1; q:=2↑i; r:=r×(q-1)/(q-2); goto L1;end;
if q=-1 then
begin B2:=true;r1:=r;if ¬B1 then begin r:=2×r;goto L2 end;end;
L3:
if xc=0∧yc=0 then begin S:=r1;goto eight circles end;
if sqrt(xc↑2+yc↑2)×tol>2×r1 then goto deflation;
eight circles:
r2:=5×r1/6; r1 := 2×r2;
for p:=1,2 do
begin comment The annulus which certainly contains at least one root is covered
with 8 circles, and the first one containing a root is chosen. If no circle can
be found, the limits of accuracy for computation has been reached and this sta-
```

```
ge of approximation must be accepted. The 4 circles with centers on the axis are
examined first next the other;
if p=1 then begin a1:=0;a2:=-r1 end else begin a1:=a3×r1;a2:=-a1 end;
for k:=1,2,3,4 do
begin
temp:=-a2; a2:=a1; a1:= temp; newxc:= xc+a1; newyc := yc+a2;
if newxc=xc∧newyc=yc then goto deflation;

comment A circle with center(newxc+inewyc) and radius r2 shall now be trans-
formed to unit circle;
r := r2; i:=1;
L4:
T(m,a,b); SYNDIV(m,a,newxc,newyc,r,true); q:=ROOT(m);
if q = 0 then
begin comment No answer can be given whether the circle in question contains
a root, the process is repeated with a greater circle;
i:=i+1; q:=3×2↑i; r:=r×(q-1)/(q-2);  goto L4
end;
if q>0 then
begin
xc:=newxc;yc:=newyc;
if q=2 then goto Newton;
r:=r/2; goto nextapprox;
end;
end k;
end p;
goto deflation;
comment end of Lehmers procedure;

Newton:
NEWTON(m,r,xc,yc,a,b,Lehmer);
NEWTON(n,r,xc,yc,a,A,Lehmer);
deflation:
SYNDIV(m,b,xc,yc,1,false);
b[m,0] := xc; b[m,1] := yc;
end m;
T(n,A,b);
end procedure LEHMERNEWTON;
```

O. Lang Rasmussen

DANISH ATOMICENERGY COMMISSION
Research Establishment Risø
Reactor Physics Department
Computer Group

September 1967
SA - 96
25 copies

Chebyshev approximation

ALGOL-procedure CHSUM

1. Scope.

The real procedure evaluates the function f(x) given as a Chebysherv approximation.

$f(x) = a[0]/2 + a[1] \times T1(x) + a[2] \times T2(y) + \ldots\ldots\ldots + a[n] \times Tn(x)$

in an interval $x1 \leq x \leq x2$.

2. Method.

The algorithm is based on a recursiv technic given by Clenshaw, see ref[1] or ref[2].

3. Use of the procedure.

The procedure call is of the type

CHSUM (n,x,x1,x2,a);

where

n: integer is the order of the Chebyshev approximation.
x: real is the independent variable for which the function f(x) shall be evaluated.
x1,x2: real is the lower respective the upper end of the interval for x.
a: array a[0:n] contains the coefficients in the Chebyshev approximation.

4. References.

[1]: C-E.Frøberg Lærobok i numerisk analys, 1962, p. 253
[2]: Modern Computing Methods sec.ed. 1961, pp 76-77.

O. Lang Rasmussen

5. Algorithm.

```
real procedure CHSUM (n,x,x1,x2,a);
value n,x,x1,x2; integer n; real x,x1,x2; array a;
begin comment this real procedure computes the value of a function f(x) given as a
            Chebyshev approximation
            f(x) = (a[0]/2) × T0(x) + a[1] × T1(x) +.....+ a[n] × Tn(x)
            in the interval x1 ≤ x ≤ x2, such as described in SA-96.;
integer i; real c,u0,u1,u2;
u2 := u1 := 0;
c := 2×(2×x-x2-x1)/(x2-x1);
for i := n step -1 until 1 do
begin
u0 := a[i]+c×u1-u2;
u2 := u1; u1 := u0;
end;
CHSUM := (a[0]+c×u1)/2-u2;
end CHSUM;
```

DANISH ATOMIC ENERGY COMMISSION
Research Establishment Risø
Electronic Department

March 1968
SA-101
25 copies

Variable integer layout

1. Scope

It is often useful printing integers, so the number of printed characters equals the number of digits in the integer plus a possible sign.

2. Method

In GIER ALGOL III layouts are of type boolean. This makes it possible to pack your own layouts, as described in A MANUAL OF GIER ALGOL III p.51.

A boolean array layout[1:n] is declared in the outermost block. n is the largest number of digits (not greater than 9) in any integer to be printed. In layout[m] is packed an integer layout with m digits ($1 \leq m \leq n$) and the appropriate sign code.

3. Use of algorithm

To print integer b, compute the number of digits in b as d:= entier(log(b))+1 (with base 10), output (layout[d],b); will print b with d digits (and a possible sign).

4. Algorithm and test

Example: After outtext(‡<p=‡); we want output of three integers with a maximum of 5 digits (n=5). The first number must be without sign (or space) if it is positive and with sign if it is negative. The other numbers are to be printed with the proper sign and no spaces.

```
begin
boolean array layout[1:10];
comment We want 2 sign codes and 5 digits wich gives 10 diffe-
        rent layouts;
integer n;
for n:= 1 step 1 until 5 do
begin
     pack(layout[n],0,39,0,20,23,n,24,27,n,34,34,1);
     pack(layout[n+5],0,39,0,20,23,n,24,27,n,28,29,2,34,34,1);
end
     layout[1:5] contains the layouts ⊄n⊅, ⊄nd⊅, ....., ⊄ndddd⊅.
     In MANUAL p.16 is shown, that alarmprinting wil insert a
minus sign in front of a negative number.
     layout[6:10] contains ⊄+n⊅, ⊄+nd⊅, ..., ⊄+ndddd⊅;
begin
integer procedure log10(a);
integer a;
comment The procedure computes the number of digits in a;
begin
log10:= if a = 0 then 1
   else entier (0.434294485×ln(abs(a)))+1
end log10;
comment The program is inserted here. A short example is shown;
integer p, r;
p:= -357;
r:=10521;
n:= -539;
outtext(⊄<p=⊅);
output(layout[log10(p)],p);
output(layout[log10(r)+5],r);
output(layout[log10(n)+5],n);
comment Note that printing of integers with more than n (in this
        case 5) digits, causes either a wrong layout or the mes-
        sage index;
end program block
end outer block;
output of this program is

     p=-357+10521-539
```

B. Runge.